De structure en structure

Activité 03

Partie 1 : retour aux basiques

Exercice 1 : dans l'épisode précédent

1. Écrire une fonction Python `prod(tab)` qui renvoie le produit des éléments d'un tableau.
2. Écrire une fonction `racines(n)` qui renvoie un tableau avec les racines carrées des entiers de 0 à `n` inclus.
3. Écrire une fonction `plus_de_a_que_de_b(texte)` qui renvoie un booléen.
4. Pour chacune des fonctions précédentes : combien d'opérations élémentaires sont nécessaires à l'exécution ?

Exercice 2 : types

En informatique, les objets utilisés ont ce qu'on appelle **un type**, tout comme, en physique, les grandeurs manipulées ont des dimensions. On distingue quatre types fondamentaux, appelés **types primitifs** :

— `int`, les entiers ;
— `float`, les flottants (i.e. les « nombres à virgules ») ;
— `bool`, les booléens ;
— `str`, les strings.

Certaines opérations ne sont possibles qu'avec certains types (par exemple la division), certaines changent de comportement selon le type (par exemple `+`).
Un type composé est obtenu lorsqu'on manipule des structures (listes, tuples, ...) et types primitifs. Par exemple, `[1,2,4,8]` est un tableau d'entiers. Techniquement, Python permet d'avoir des objets de types différents stockés dans un tableau, mais cela hérisse le poil des informaticiens, donc on évitera cette année.
Déterminer le type des objets suivants :

1. `1456;`
2. `1456.2;`
3. `1456.0;`
4. `"bonjour !";`
5. `"A";`
6. `'a';`
7. `"'";`
8. `["a","b","c","d"];`
9. `'["a","b","c","d]';`
10. `[("Nadal",14),("Borg",6),("Cochet",4)]`
11. `[[1,2,3],[4,5,6]]`
12. `["a",'b',["c","d"]]`

Exercice 3 : signature

Les fonctions aussi ont des types, ou plus précisément, des **signatures**. Considérons la fonction suivante :

$$\phi : \begin{pmatrix} \mathbb{Q} & \to & \mathbb{R} \\ x & \mapsto & \sqrt{x} \end{pmatrix}$$

La fonction ϕ va des rationnels vers les réels : sa signature (attention, ça n'existe pas en mathématiques ...) serait $\phi : \mathbb{Q} \to \mathbb{R}$: quel type en entrée ? quel type en sortie ? Par exemple, la fonction `sqrt` du module `math` est de signature `float -> float`.
Déterminer les signatures des fonctions suivantes.

```python
def f1(x) :
    return 10%x
```

```python
def f2(x) :
    return len(x)
```

```python
def f3(x,y):
    return x//y
```

```python
def f4(x) :
    return x or x
```

```python
def f5(x) :
    s = 0
    for i in range(len(x)) :
        s += x[0]
    return s
```

```python
def f6(x) :
    if x == [] :
        print("Tableau vide !")
    else :
        return len(x)
```

```python
def f7(x) :
    if x == [] :
        return 0
    else :
        c = 0
        for i in range(len(x)) :
            print(x[i])
            c+=1
        return c
```

Exercice 4

Une structure importante, que nous reverrons incessamment, est celle de **chaîne de caractères** (ou string). Les strings servent spécifiquement à stocker du texte, et ressemblent à `"Bonjour, je suis du texte."` (guillemets doubles) ou `'youpi youpi youpi'` (guillemets simples).

1. Tester les commandes suivantes.
2. Les strings sont-ils mutables ?
3. Écrire une fonction `est_dedans(lettre,texte)` qui renvoie `True` si `lettre` apparaît dans `texte`, et `False` sinon.
4. Écrire une fonction `begayant_string(s)` qui renvoie le bégayant du string `s` sous forme d'un string : par exemple, `begayant_string("oui")` renverra `"oouuii"`.

```
>>> str = "J'adore l'informatique"
>>> str
>>> str[3]
>>> len(str)
>>> tup[2] = "o"
>>> tup + " et les mathématiques"
```

5. Écrire une fonction `sans_la_fin(texte)` qui renvoie `texte` privé de son dernier caractère.

Exercice 5

Écrire en Python une fonction `nb_voyelles(str)` qui renvoie le nombre de voyelles dans le string `str` (dont on supposera toutes les lettres en minuscule).

Exercice 6

1. Écrire une fonction `lettres(texte)` qui renvoie le tableau des caractères apparaissant dans `texte`. Par exemple, `lettres("abracadabra")` renverra `["a","b","r","c","d"]`.
2. Écrire une fonction `lettre_la_plus_commune(texte)` qui renvoie ... c'est dans le titre.
3. Écrire une fonction `mots(texte)` qui renvoie le tableau des mots présents dans le string `texte`. On suppose que les mots sont séparés par des espaces (et aucun autre signe de ponctuation). Par exemple, `mots("Bonjour j'aime beaucoup l'informatique")` renverra `["Bonjour","j'aime","beaucoup","l'informatique"]`.
4. On dit qu'un mot `m1` est une anagramme du mot `m2` s'il est possible de réarranger les lettres de `m1` pour former le mot `m2`. Écrire une fonction Python `est_anagramme(m1,m2)` qui renvoie `True` si `m1` est une anagramme de `m2`.

Partie 2 : plus de structures

Exercice 7

Un **dictionnaire** est une structure très similaire, mais plutôt que d'avoir des indices entiers, les éléments sont étiquetés par ce qu'on veut, par exemple des strings : `{'bonjour': 'hello', 'au revoir': 'goodbye', 'merci': 'thank you'}`. Dans un dictionnaire, on ne parle plus d'indice, mais de **clé**, à laquelle on associe une **valeur**. Bien que les dictionnaires semblent résoudre beaucoup de problèmes en informatique, leur implémentation en machine est en fait loin d'être triviale, et rend les opérations dessus non triviales. Pour le moment, nous fermerons les yeux sur ces difficultés : on les traitera comme des tableaux.

1. Initialiser une variable `dico` au dictionnaire vide, noté `{}`.
2. Ajouter une case à ce dictionnaire : `dico["a"] = 1`. Qui est la clé ? la valeur ?
3. Demander la valeur associée à la clé `"a"`, même syntaxe que les listes.
4. Vérifier si `"a"` est une clé de `dico` : `"a" in dico`.
5. Modifier la valeur associée à la clé `"a"` : `dico["a"] = 4`.

Exercice 8

1. Écrire une fonction `nb_occurrences(tab) : list -> dict` qui prend en paramètre un tableau `tab`, et renvoie un dictionnaire fournissant le nombre d'occurrences des éléments de `tab`. Par exemple, si `d = nb_occurrences(["a","b","c","a","d","c","a"])`, alors `d["a"] = 3` et `d["c"] = 2`.
2. Écrire une fonction `moyenne_occurrences(tab) : list -> float` qui renvoie la moyenne des nombres d'occurrences des éléments à partir de `nb_occurrences`. La syntaxe « `for k in d` » permet de parcourir les clés (`k`) dans le dictionnaire `d`. Déterminer un autre algorithme plus efficace pour obtenir la même quantité.

Exercice 9

Écrire une fonction `scrabble(mot) : string -> int` qui renvoie le score d'un mot au Scrabble français. Pour rappel, les valeurs des lettres sont les suivantes :

K,W,X,Y,Z	J,Q	F, H, V	B, C, P	D, G, M	le reste
10	8	4	3	2	1

On créera un dictionnaire pour ce faire. En admettant que la création du dictionnaire en question est négligeable, combien d'opérations élémentaires sont nécessaires pour exécuter `scrabble(mot)` en fonction de la longueur de `mot` ?

Exercice 10

En Python, lorsqu'on a un tableau `tab`, il peut être utile d'utiliser des **tranches** (ou *slices*) : c'est simplement l'idée de demander un sous-tableau. Par exemple, si on a un tableau `tab = ["a","b","c","d","e","f"]`, la notation `tab[2:5]` représentera le sous-tableau constitué des cases 2 (inclus) à 5 (exclus), dans notre cas `["c","d","e"]`. Cette notation fonctionne aussi sur les strings et les tuples.

1. Créer une variable `texte = "abcdefghi"`.
2. Tester les commandes suivantes :
   a) `texte[3:7];`
   b) `texte[3:3];`
   c) `texte[:5];`
   d) `texte[2:];`
   e) `texte[-1];`
   f) `texte[2:-1].`
3. Un mot `fac` est un **facteur** d'un autre mot `mot` si `fac` apparaît comme « sous-bloc » de `mot` : par exemple, `"stuc"` est un facteur de `"lustucru"`. À l'aide des tranches, écrire une fonction Python

`est_facteur(fac,mot) : (string * string) -> bool` qui renvoie `True` si `fac` est un facteur de `mot`. *Attention : la notion de « sous-mot » est une notion différente de celle de facteur. Dans le cas des sous-mots, on autorise à ce que les lettres ne soient pas d'un seul tenant dans `mot`. Par exemple, `"uuu"` est un sous-mot de `"lustucru"`.*

4. L'objectif est d'écrire une autre version de la fonction précédente sans utiliser la notion de tranche.
   a) Écrire une fonction `est_facteur_emplacement(fac,mot,i): (string*string*int) -> bool` qui renvoie `True` si `fac` apparaît dans `mot` à partir de la position `i`, et `False` sinon.
   b) Écrire une fonction `est_facteur_2(fac, mot)` à partir de la fonction précédente.
5. On note `n` la longueur de `mot` et `m` la longueur de `m` : combien d'opérations élémentaires effectue-t-on **au maximum** ? (par ailleurs : à quoi correspond ce cas maximal ?)

Exercice 11

En Python, on représente les matrices comme des tableaux de tableaux : par exemple `[[1,2,3],[4,5,6]]` représente la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

1. Créer une variable `t = [[1,2,3],[4,5,6]]`. Quel est son type ? À quoi correspond `t[0]` ? `t[0][1]` ?
2. Écrire une fonction `ligne(tab,i)` et `colonne(tab,j)` qui renvoie la `i`-ème ligne ou `j`-ème colonne d'une matrice représentée par `tab`.

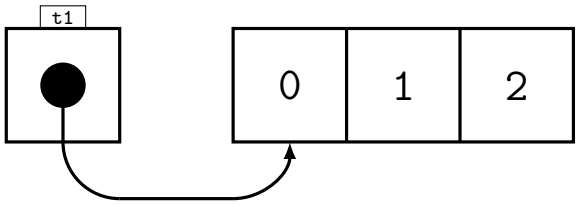
Exercice 12

En Python, la gestion des tableaux n'est pas si simple qu'on ne le croie.

1. **Sur papier**, exécuter le code ci-contre de manière « intuitive ». En particulier, préciser la valeur affichée à la dernière ligne.
2. Tester le code sur machine : est-ce le résultat auquel vous vous attendiez ?

   Pour expliquer ce phénomène, il faut introduire la notion de **pointeur**. Dans la mémoire de l'ordinateur, un tableau `tab` n'est pas directement une succession de cases : c'est en fait une boîte contenant une flèche qui dirige vers une succession de cases.

```
t1 = [0,1,2]
t2 = t1
t2[0] = 10
print(t1[0])
```



Quand on écrit `t2 = t1`, on procède à une copie de la flèche : `t2` va donc pointer vers la même case. Et désormais, si on modifie la case pour `t2`, la case vers laquelle `t1` pointe est aussi modifiée.

3. Décrire l'intégralité du problème à l'aide de schémas boîte.
4. Écrire une fonction `copie(t) : list -> list` qui renvoie une copie décorrélée de `t` dans un nouveau tableau, et tester : exécuter `s = t` puis `s[0] = 19` devrait ne pas changer la valeur dans `t`.

Exercice 13

Dans cet exercice, on considère des tableaux de couples (par exemple `[(1,2),(0,4),(10,7)]`).

1. On dit qu'un tableau est croissant si tous les couples du tableau `(x,y)` vérifient $x \leqslant y$. Écrire une fonction `est_croissant(tab) : (int*int) list -> bool` qui renvoie `True` si le tableau est croissant.
2. Un tableau est dit alternant si les couples alternent entre croissance et décroissance. Par exemple, `[(2,4),(5,1),(-2,2),(10,4)]` est alternant : le premier couple croît, le second décroît etc. Écrire une fonction `est_alternant(tab) : (int*int) list -> bool` qui renvoie `True` si le tableau est alter-

nant. Attention, un tableau alternant peut commencer par une croissance ou une décroissance.

3. Un tableau est dit convexe si la différence entre les éléments des couples croît. Par exemple, `[(2,0),(3,2),(5,5),(0,2),(-5,0)]` est convexe : lorsqu'on calcule la différence à l'intérieur de chaque couple, on obtient respectivement −2, −1, 0, 2, 5, et ces différences sont croissantes. Écrire une fonction `est_convexe(tab) : (int*int) list -> bool` qui renvoie `True` si le tableau `tab` est convexe.

Exercice 14

1. Écrire une fonction `est_trie(tab) : list -> bool` qui renvoie `True` si le tableau `tab` est trié dans l'ordre croissant.
2. On va écrire une fonction de tri : elle renverra la version triée d'un tableau. On va plus précisément implémenter l'algorithme de tri le plus simple (mais pas le plus efficace) : le tri par sélection.

   L'algorithme du tri par sélection est l'algorithme de pseudo-code suivant :

   Données : un tableau `tab`
   $n \leftarrow$ longueur de `tab`
   pour $i = 0$ **à** $n - 1$ **:**
   $\quad j \leftarrow$ indice du minimum de `tab[i:]`
   $\quad$ échanger les cases `tab[i]` et `tab[j]`
   retourner `tab`

3. Exécuter le tri par sélection sur le tableau `[12,11,14,13,10]`.
4. Écrire une fonction `indice_minimum(tab,i) : (list,int) -> int` qui renvoie l'indice du minimum de `tab` à droite de `i` inclus. Par exemple, `indice_minimum([12,11,14,13,10],2)` renverra 4 : le minmum à droite de 2 est 10, situé à l'indice 4.
5. Écrire une fonction `tri_selection(tab) : list -> list` qui renvoie le tableau `tab` trié.

Pour la prochaine fois :

— Pour un tableau `tab` de longueur `n`, on dit que `(i,j)` est un plateau de `tab` si `tab[i:j]` est constant. Par exemple, `[9,2,2,2,4,3,3,1]` admet `(5,7)` comme plateau (correspondant aux 3) dans le tableau. Écrire une fonction `plus_long_plateau(tab) : list -> (int*int)` qui renvoie le plus long plateau de `tab`.

— Écrire une fonction `est_trie_couples(tab) : (int*int) list -> bool` qui renvoie `True` si un tableau de couples est trié en « découplant » les couples.

Par exemple, `[(1,2),(3,4),(5,6)]` est trié, mais pas `[(1,4),(2,5),(3,6)]`.

— Une matrice est un tableau comme `[[1,2,3],[4,5,6],[7,8,9]]`, dont les cases contiennent elles-mêmes des tableaux. Écrire une fonction `aplatir(matrice)` qui renvoie la version « aplatie » de matrice : par exemple, `aplatir([[1,2,3],[4,5,6],[7,8,9]])` renverra `[1,2,3,4,5,6,7,8,9]`.